UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Aceleración de un motor FIX mediante DPDK para mejorar la rentabilidad de estrategias que operan en FOREX

Luis Aguilera Hernández
Tutor: Juan Antonio Andrés Sáez
Ponente: Jorge Enrique López de Vergara Méndez

JUNIO 2019

Aceleración de un motor FIX mediante DPDK para mejorar la rentabilidad de estrategias que operan en FOREX

AUTOR: Luis Aguilera Hernández
TUTOR: Juan Antonio Andrés Sáez

Resumen

Este Trabajo de Fin de Grado consiste en la modificación de un motor FIX con el fin de acelerar su funcionamiento mediante el uso de una tecnología reciente de Intel llamada DPDK (*Data Plane Development Kit).*

Un Motor FIX es un programa capaz de conectarse a los mercados financieros e interactuar con ellos en tiempo real haciendo uso del protocolo FIX. Se trata de un programa que se comunica con el mercado y realiza transacciones en cuestión de microsegundos. Hoy en día en los mercados financieros se está empezando a operar con estrategias diseñadas mediante algoritmos automáticos que analizan constantemente los precios y realizan miles de operaciones por segundo. En este contexto, la reducción del tiempo que se tarda en realizar alguna operación de compra o venta, o simplemente en el análisis de información de precios, supone una gran ventaja de cara al resto de estrategias independientemente de la efectividad de los algoritmos internos de las mismas.

Dado que la pila TCP/IP del sistema operativo original no esta diseñada para entornos de muy baja latencia, para conseguir una latencia menor, no solo haremos uso de DPDK, vamos a utilizar una pila TCP/IP llamada ANS (*Accelerated Network Stack)* que hace uso de DPDK y nos proporciona las facilidades necesarias para conectarnos con un cliente FIX. Este cliente se va a desarrollar en el lenguaje de programación C haciendo uso de la librería *libtrading* para facilitarnos su implementación con respecto al protocolo FIX.

El objetivo final es disminuir la velocidad de procesamiento del motor FIX, demostrando así que una reducción considerable en la latencia a nivel de Sistema Operativo y motor FIX mejoran el rendimiento de las diferentes estrategias que se pueden aplicar en el mercado de FOREX.

Palabras clave

DPDK, ANS, hipervisor, FIX, FOREX, anfitrión, huésped, modo puente, Interfaz de Programación de Aplicaciones, núcleo, banco de pruebas.

Abstract

This Bachelor Thesis consists of the modification of a FIX engine in order to accelerate its operation through the use of a recent Intel technology called DPDK (Data Plane Development Kit).

A FIX Engine is a program capable of connecting to financial markets and interacting with them in real time using the FIX protocol. It is a program that communicates with the market and makes transactions in a matter of microseconds. Today's financial markets are starting to operate with strategies designed by automatic algorithms that constantly analyze prices and perform thousands of trades per second. In this context, the reduction of the time it takes to carry out a purchase or sale operation, or simply in the analysis of price information, is a great advantage over other strategies regardless of the effectiveness of the internal algorithms of the same.

Since the TCP/IP stack of the original operating system is not designed for very low latency environments, to achieve a lower latency, we will not only use DPDK, we will use a TCP/IP stack called ANS (Accelerated Network Stack) that makes use of DPDK and provides us with the necessary facilities to connect to a FIX client. This client will be developed in the programming language C using the libtrading library to facilitate its implementation with respect to the FIX protocol.

The final objective is to reduce the processing speed of the FIX engine, thus demonstrating that a considerable reduction in latency at the level of Operating System and FIX engine improve the performance of the different strategies that can be applied in the FOREX market.

Keywords

DPDK, ANS, hypervisor, FIX,FOREX, host, guest, bridge mode, Aplication Programming Interface (API), kernel, testbed.

Agradecimientos

En primer lugar, me gustaría dar las gracias a mi tutor, Juan Antonio Andrés Sáez, por darme la oportunidad de realizar este trabajo y guiarme a lo largo de su desarrollo. También me gustaría agradecérselo a mi ponente Jorge López de Vergara por hacer esto posible.

Sobre todo quiero dar las gracias a mi familia. A mis padres, por su apoyo y paciencia. Y gracias a todos mis seres queridos sin los que tanto este trabajo como la carrera no habrían sido iguales.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

Glosario

- API (Application Programming Interface): Es un conjunto de subrutinas, funciones y procedimientos, que ofrece cierta biblioteca para ser utilizado por otro *software* como una capa de abstracción.
- Ask y Bid: Son los precios de demanda y oferta, es decir, los que determinan la cotización de un valor. **Bid** es el precio más alto que el comprador está dispuesto a pagar, y **ask** el precio más bajo al que el vendedor está dispuesto a vender.
- Banco de pruebas (testbed): es una plataforma para experimentación de proyectos de gran desarrollo. Los bancos de pruebas brindan una forma de comprobación rigurosa, transparente y repetible de teorías científicas, elementos computacionales, y otras nuevas tecnologías. Es un método para probar un módulo particular (función, clase, o biblioteca) en forma aislada.
- Forex (Foreign Exchange): Se le conoce como mercado de divisas. En el se compran y venden divisas de todo el mundo.
- Hipervisor: Es una plataforma que permite aplicar diversas técnicas de control de virtualización para utilizar, al mismo tiempo, diferentes sistemas operativos (sin modificar o modificados, en el caso de paravirtualización) en una misma computadora.
- Mercado financiero: Es un espacio en el que se realizan los intercambios de instrumentos financieros y se definen sus precios.
- Latencia: Se define como el tiempo que ocurre entre que envías una petición hasta que recibes el primer bit de respuesta. En redes informáticas de datos la latencia es la suma de retardos temporales dentro de una red. Un retardo es producido por la demora en la propagación y transmisión de paquetes dentro de la red.

1 Introducción

1.1 Motivación

En un mundo en el que la red se está convirtiendo en fundamental para las comunicaciones, el rendimiento y la latencia son cada vez mas importantes. Esto junto con el aumento constante de las capacidades de cálculo de los computadores, hace que cada vez sea más frecuente que los traders operen haciendo negociaciones de alta frecuencia, o conocido por sus siglas en ingles HFT (High-Frequency Trading).

Este proceso de inversión se lleva a cabo mediante herramientas tecnológicas sofisticadas para obtener información del mercado y en función de esta intercambiar valores. Cada posición de inversión se mantiene durante breves periodos de tiempo (fracciones de segundo) para rápidamente comprar o vender según interese, por lo que se hacen decenas de miles de operaciones al día.

Se trata de un tipo de inversión muy sensible a la velocidad de procesamiento de los datos del mercado y al tiempo que tardan estos en enviarse y recibirse entre las distintas partes y contrapartes. Precisamente esta sensibilidad al tiempo de procesamiento es lo que ha motivado la realización de este TFG en torno a la optimización de dicho tiempo y el análisis de sus consecuencias.

1.2 Objetivos

El objetivo principal de este TFG es reducir la latencia en el procesamiento de ordenes de los mercados financieros y demostrar que dicha optimización beneficia a las diferentes estrategias de trading que se pueden emplear.

Para llegar a cumplir el objetivo principal, primero deberemos llegar a objetivos más segmentados, como son:

- Configurar el entorno y las herramientas necesarias para ejecutar correctamente el DPDK de Intel.
- Configurar el entorno y las herramientas para ejecutar ANS sobre DPDK.
- Familiarizarse y realizar las modificaciones necesarias en el código abierto de ANS y las librerías de *libtrading* para lograr una integración eficiente y poder utilizarlas conjuntamente.

- Desarrollar un cliente FIX que haga uso de todas estas librerías, sea capaz de operar independientemente en el mercado, y permita trabajar indistintamente con la pila TCP/IP nativa del S.O. y también con ANS de modo que ambas pilas puedan ser comparadas a partir de un mismo testbed.

1.3 Organización de la memoria

1.3.1 Fases de realización del TFG

Tareas a realizar:

1. Documentación del trabajo. Se lleva a cabo durante todo el desarrollo del TFG. Se ha documentado todo lo realizado y relacionado con el TFG .
2. Instalación y configuración del entorno de trabajo. Se realiza la instalación de las maquinas virtuales, con los requerimientos necesarios, y el sistema operativo Fedora, así como la configuración del host en modo bridge.
3. Instalación y configuración de DPDK. En esta tarea se ha realizado primero la instalación de DPDK con sus diversas especificaciones. Después se ha realizadola configuración del entorno de DPDK. Es importante destacar que en esta fase ha habido diversos retrasos debido a los errores surgidos a la hora de instalar DPDK, aunque finalmente se consiguieron resolver.
4. Instalación y configuración de ANS. Se ha instalado la librería de ANS junto con DPDK configurando conjuntamente ambas librerías.
5. Instalación e integración de *libtrading*. En esta tarea se han integrado las librerías de *libtrading* junto con ANS y DPDK para su uso conjunto.
6. Programación del cliente FIX. A lo largo de esta tarea se ha programado el cliente FIX tanto a nivel de sockets como de las especificaciones exigidas por el protocolo FIX. Se han desarrollado dos clientes, uno integrado con las librerías de ANS y DPDK, y otro sin ellas.
7. Pruebas y resultados. En esta tarea se han realizados las pruebas con los dos clientes desarrollados previamente y un servidor externo, y se han obtenido los resultados de la optimización de los tiempos.

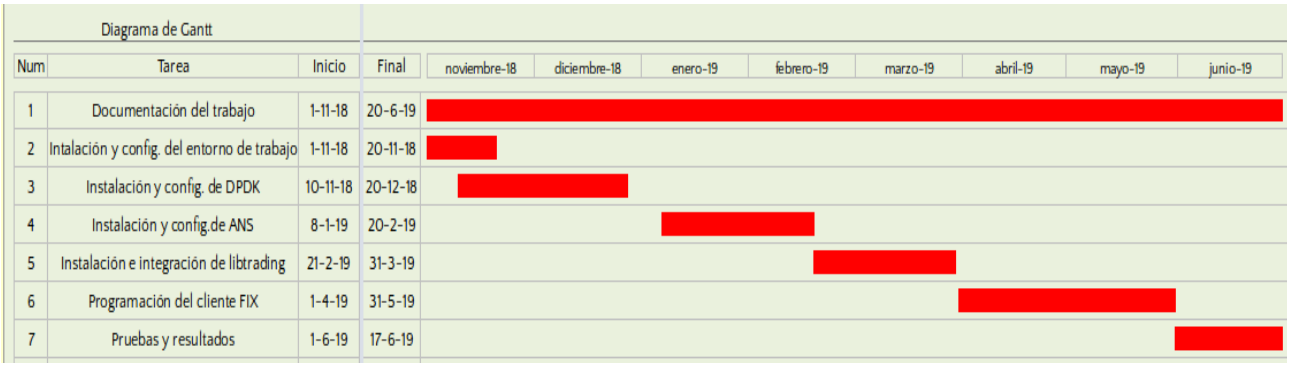

Figura 1-1: Diagrama de Gantt de todo el desarrollo del TFG

1.3.2 Estructura de la memoria

La memoria consta de los siguientes capítulos:

- **Estado del arte.** En este capítulo se realiza una pequeña descripción general de la situación actual de las tecnologías que se han usado a lo largo del desarrollo de este TFG.
- **Diseño.** En este capítulo se describirán la configuración y las características del entorno de trabajo en el que se ha desarrollado este TFG.
- **Desarrollo.** En este capítulo veremos paso a paso como se han implementado las diferentes fases del TFG hasta llegar a los resultados obtenidos.
- **Pruebas y resultados.** En este capítulo se explicarán las diversas pruebas realizadas y los resultado correspondientes obtenidos.
- **Conclusiones y trabajo futuro.** En este capítulo se resumen las conclusiones extraídas y se analizan brevemente las posibilidades futuras de investigación y desarrollo en diferentes líneas, a partir de los resultados obtenidos de este TFG.

2 Estado del arte

2.1 Intel Data Plane Development Kit (DPDK)

DPDK son una serie de librerías y controladores Open Source relativamente nuevas y desarrolladas por Intel. Se encargan de acelerar las cargas de trabajo del procesamiento de paquetes de red [1]. Sus características mas importantes son:

- Se pueden ejecutar en una amplia variedad de arquitecturas CPU.
- El procesamiento de paquetes se realiza vía hardware, lo cual favorece la reducción del tiempo.
- Recepción y envío de paquetes dentro del número mínimo de ciclos de CPU.
- La configuración se realiza desde el nivel de usuario, mediante la API de la librería.

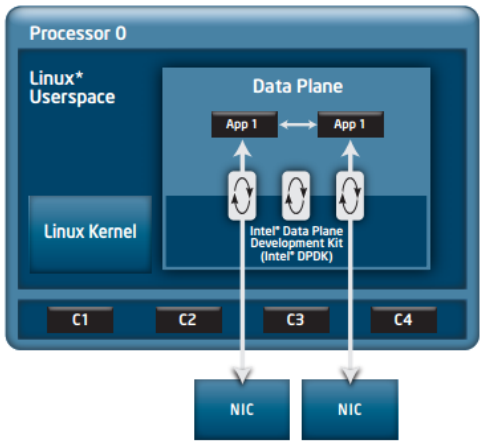

Figura 2-1: Esquema del funcionamiento de DPDK

2.2 ANS (Accelerated Network Stack)

Es una pila TCP/IP Open Source en el espacio de usuario que utiliza también las librerías de DPDK. Nos va a permitir conectar DPDK con cualquier aplicación que queramos desarrollar encima de esta pila [9].

Se trata de una pila TCP/IP de baja latencia con una API que permite configurar diferentes parámetros mediante diversos comandos desde el espacio de usuario de forma inmediata e intuitiva.

2.3 Libtrading

Libtrading es una API de código abierto para aplicaciones de trading de alto rendimiento y baja latencia. Implementa protocolos de red utilizados para la comunicación con bolsas, fondos de inversión y otros lugares de negociación. La API soporta FIX, FIX/FAST, y muchos protocolos propietarios como ITCH y OUCH utilizados por NASDAQ [11].

Estas librerías nos van a proporcionar las funciones y las estructuras de código necesarias para establecer las especificaciones del protocolo FIX e incluso añadir algunas nuevas de forma manual, según los requerimientos que necesitemos.

2.4 QEMU/KVM

QEMU es un emulador genérico y de código abierto de máquinas virtuales. Cuando se utiliza como virtualizador, QEMU alcanza un rendimiento cercano al nativo, ejecutando el código de invitado directamente en el CPU host. Puede usar hipervisores como Xen o KVM [5].

KVM, Kernel-based Virtual Machine, es un hipervisor integrado en el núcleo de Linux. A diferencia del QEMU nativo, que utiliza emulación, KVM es un modo operativo especial de QEMU que utiliza extensiones de CPU (HVM) para la virtualización mediante un módulo del núcleo [4].

Usando KVM, se pueden ejecutar múltiples máquinas virtuales que ejecuten GNU/Linux sin modificar, Windows o cualquier otro sistema operativo. Cada máquina virtual tiene hardware virtualizado privado: una tarjeta de red, un disco, una tarjeta gráfica, etc.

2.5 Fedora

Fedora es una distribución Linux de código abierto, que incluye software libre. Se caracteriza por ser estable y mantenida gracias a una comunidad de ingenieros, desarrolladores gráficos y usuarios que informan de fallos y prueban nuevas tecnologías [2]. Fedora es un sistema operativo fiable, potente y fácil de usar para equipos portátiles y de escritorio. Es funcional para una amplia gama de desarrolladores, desde aficionados y estudiantes hasta profesionales en entornos empresariales.

2.6 Financial Information eXchange (FIX)

El protocolo FIX es una especificación de formatos de mensajes para la comunicación electrónica de información financiera. Ha sido desarrollado a través de la colaboración de bancos, agentes de bolsa, mercados y otras entidades financieras de todo el mundo, que crearon la asociación FPL (Fix Protocol Limited) a fin de mantener y promocionar su uso ante las necesidades cambiantes de los mercado financieros globales.

FIX permite que dos entes independientes, con sistemas desarrollados en forma separada, puedan intercambiar mensajes de administración de órdenes (alta, baja, etc), información de concertación e información de mercado en forma estandarizada, con el único requerimiento de que las dos partes “hablen” el protocolo FIX.

3 Diseño

En este capítulo vamos a ver todo lo relacionado con el diseño y la configuración del entorno de trabajo en el que se ha desarrollado este TFG.

3.1 Diseño del entorno de trabajo.

Debido a las especificaciones de hardware que se requerían para poder utilizar DPDK, y a la falta de disposición de las mismas, se ha optado por desarrollar este TFG en un entorno virtualizado que nos permita hacer uso del hardware requerido, de una forma virtualizada, y ejecutar correctamente esta tecnología de Intel.

Cabe destacar que el TFG ha sido pensado para poder trabajar en un medio virtualizado o real. De este modo, el desarrollo principal y sus correspondientes pruebas se han hecho en el entorno virtualizado. Y para las pruebas reales es necesario utilizar un entorno no virtualizado con hardware soportado por las diferentes librerías, sin necesidad de cambiar prácticamente nada de la estructura desarrollada en el entorno virtualizado.

Como sistema operativo del equipo anfitrión se utiliza Ubuntu 16.04, y como sistema operativo del equipo/os huésped se utiliza Fedora 28.

Para llevar a cabo la virtualización, en un principio se hizo uso de VMware Player, sin embargo, surgieron problemas con este software de virtualización ya que no proporcionaba las opciones necesarias para asignar el hardware que se requería a la hora de instalar y compilar DPDK. Finalmente, se hizo uso de QEMU/KVM, que nos ha permitido virtualizar sobre Ubuntu 16.04 el sistema operativo Fedora 28, con todos los requerimientos necesarios.

El esquema de configuración que se ha utilizado para la realización de pruebas y el desarrollo ha sido:

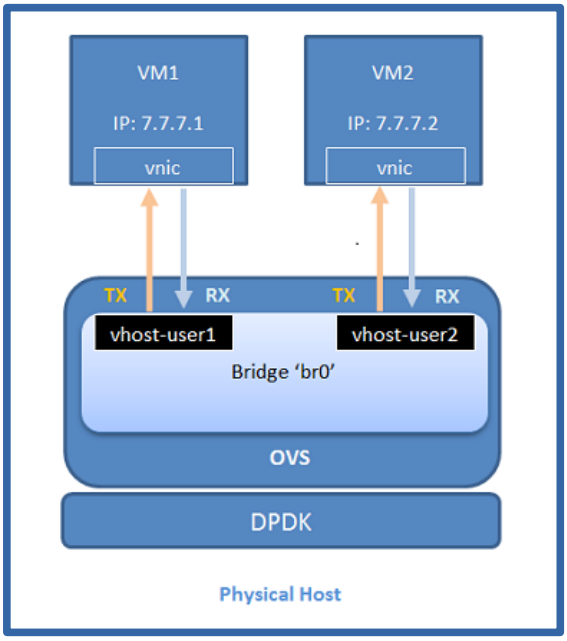

Figura 3-1: Esquema del entorno de trabajo. Host con SO Ubuntu 16.04 y máquinas virtuales con Fedora 28

3.1.1 Configuración del Host.

Como se ha dicho antes, debido a la falta de recursos (hardware en este caso), se decidió desarrollar este TFG en un entorno virtualizado. Esto nos ha permitido no solo disponer del hardware necesario, como tarjetas de red específicas, sino de otros ordenadores (virtuales) para simular una red cliente-servidor.

Para llevar a cabo la configuración del esquema de trabajo mostrado en la Figura 3-1, es necesario configurar nuestro equipo anfitrión en modo "bridge". El modo bridge hace que la máquina virtual se comporte como una nueva máquina física. Estos dispositivos en realidad se comunicarán con la red por la única tarjeta física existente pero permitirán, por un lado, acceder al ordenador mediante varias IPs, cada una correspondiendo a uno de los dispositivos virtuales, y por otro asignar tareas específicas a cada uno de esos dispositivos.

Todo esto nos permitirá conectarnos entre equipos, ya sean virtuales o no, de forma independiente.

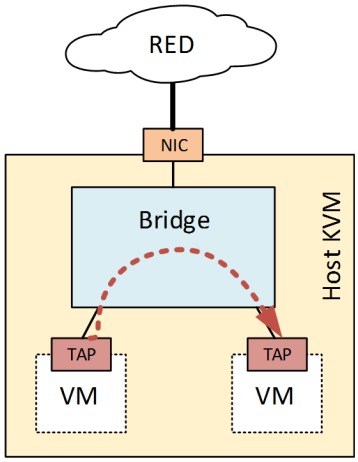

Figura 3-2: Esquema de la conexión del Host y las VM

3.1.2 Configuración de la Máquina Virtual.

Teniendo configurado el equipo en modo bridge, ahora es necesario crear y configurar las maquinas virtuales que vamos a utilizar. Una vez hemos descargado e instalado todos los paquetes de QEMU y KVM necesarios para la virtualización, haremos uso de un gestor de máquinas virtuales, que nos permitirá configurar de forma fácil y gráfica todas las características de nuestra máquina virtual.

Sin olvidar que como sistema operativo estamos usando Fedora, el aspecto más relevante a la hora de configurar nuestra máquina virtual es la tarjeta de red, tanto el modelo como el modo de conexión. Habiendo configurado previamente el host en modo bridge, el gestor de máquinas virtuales nos permite configurar la Interfaz de Red Virtual seleccionando como fuente de red el puente creado ‘br0’ y como modelo de dispositivo ‘virtio’. Este último es un requisito para que la máquina soporte DPDK y lo podamos ejecutar sin inconvenientes.

3.2 Estructura completa del programa

La idea general consiste en crear una pila de protocolos. Esto es, una colección ordenada de protocolos organizados en capas que se ponen unas encima de otras y en donde cada protocolo implementa una abstracción encuadrada en la abstracción que proporciona la capa sobre la que está encuadrada. Los protocolos encuadrados en la capa inferior proporcionan sus servicios a los protocolos de la capa superior para que estos puedan realizar su propia funcionalidad.

Como referencia tenemos el **modelo de interconexión de sistemas abiertos (OSI).** Este modelo especifica el protocolo que debe usarse en cada capa. Hay que destacar que se trata de un *modelo*, es decir, una construcción teórica. Se trata de una normativa estandarizada útil para entender el sistema de comunicación de redes, independientemente de las tecnologías utilizadas, sobre todo cuando hablamos de la red de redes, es decir, Internet.

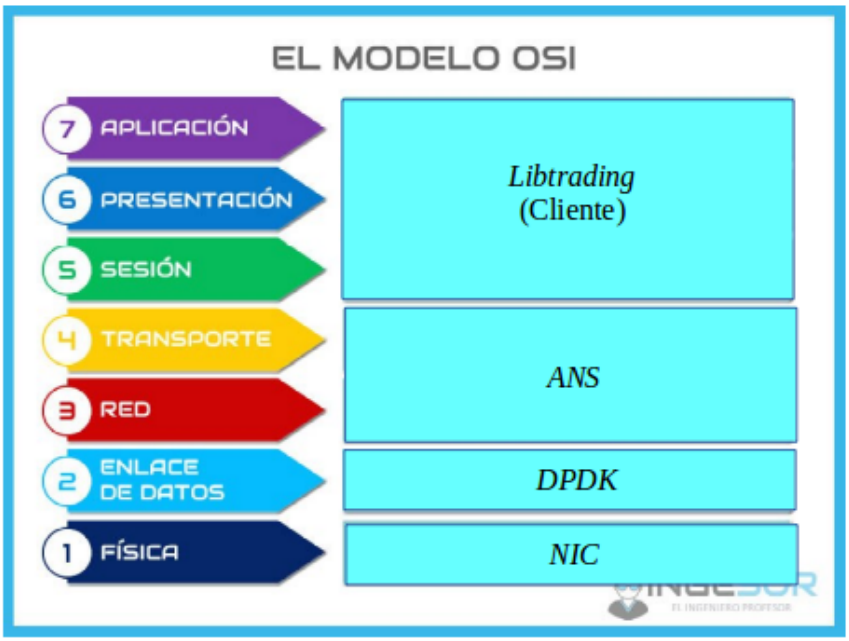

Figura 3-3: Integración de nuestra pila de protocolos en el modelo OSI

Como podemos apreciar en la Figura 3-3, a lo largo del desarrollo de este TFG se ha creado una pila de protocolos optimizada, integrando las capas en orden ascendente según el modelo OSI.

Según se vaya explicando el desarrollo del trabajo se irá haciendo referencia a esta pila de protocolos para tener claro y estructurado en que fase de la integración de capas nos encontramos.

4 Desarrollo

En este capítulo se aborda la descripción de las diferentes etapas del desarrollo del cliente FIX. Se explica como se han ido montando las diferentes capas, partiendo desde el nivel más bajo (físico, enlace..) hasta el más alto (aplicación), necesarias para la optimización y adaptación del cliente.

4.1 DPDK

4.1.1 Introducción

DPDK fue creado en el año 2010 por Intel y se hizo disponible bajo una licencia de código abierto. La comunidad de código abierto se estableció en DPDK.org en 2013 por 6WIND y ha facilitado la expansión continua del proyecto. Desde entonces, la comunidad ha estado creciendo continuamente en términos del número de contribuyentes, parches y organizaciones contribuyentes, con 5 versiones principales completadas, incluyendo las contribuciones de más de 160 personas de 25 organizaciones diferentes. DPDK ahora soporta todas las principales arquitecturas de CPU y NICs de múltiples proveedores, lo que lo hace ideal para aplicaciones que necesitan ser portátiles a través de múltiples plataformas [1].

DPDK corresponde a la capa de enlace de datos, y a continuación, vamos a explicar más en detalle su funcionamiento, los pasos que hay que seguir para poder utilizar este set de librerías y controladores y los requisitos necesarios que debe tener nuestro ordenador. Cabe destacar que todo lo que vamos a mencionar, se trata desde el punto de vista de un entorno virtualizado. Aunque esto no afecte en gran parte, si que habrá algunos pasos y requerimientos diferentes a los que se exigirían si se tratase de un entorno real.

4.1.2 Funcionamiento

DPDK está diseñado para funcionar en procesadores x86, POWER y ARM, se ejecuta principalmente en Linux, con un puerto FreeBSD disponible para un subconjunto de funciones de DPDK, y está bajo la Licencia BSD de Código Abierto.

La estructura DPDK crea un conjunto de librerías para entornos de hardware/software específicos mediante la creación de una capa de abstracción de entorno (EAL). EAL es responsable de obtener acceso a recursos de bajo nivel tales como hardware y espacio de memoria. Oculta las especificaciones del entorno y proporciona una interfaz de

programación estándar a las librerías, aceleradores de hardware disponibles y otros elementos de hardware y sistemas operativos (Linux, FreeBSD). Es responsabilidad de la rutina de inicialización decidir cómo asignar estos recursos (es decir, espacio de memoria, dispositivos, temporizadores, consolas, etc.).. Una vez creado el EAL para un entorno específico, los desarrolladores se vinculan a la biblioteca para crear sus aplicaciones. Por ejemplo, EAL proporciona los frameworks para soportar Linux, FreeBSD, Intel IA-32 o 64-bit, IBM POWER9 y ARM 32- o 64-bit [7].

DPDK también incluye ejemplos de software que destacan las mejores prácticas para la arquitectura de software, consejos para el diseño y almacenamiento de la estructura de datos, utilidades de perfilado y ajuste del rendimiento de aplicaciones y consejos que abordan los déficits de rendimiento de red más comunes.

DPDK es un framework que se utiliza para el procesamiento rápido de paquetes. Accede a la NIC directamente desde el espacio de usuario, evitando así el kernel y su sobrecarga. Como vemos en esta imagen, algunas de las ventajas de la utilización de DPDK son:

1. Se elimina la copia de datos del kernel al espacio de usuario.
2. No tiene interrupciones.
3. La pila de red se puede optimizar y agilizar.

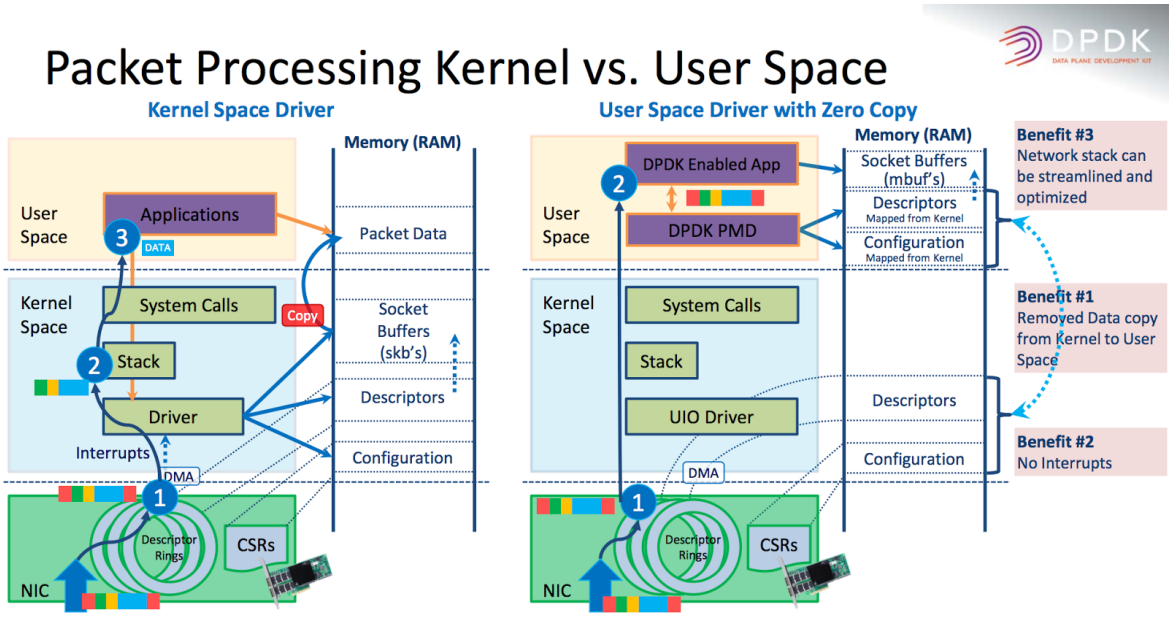

Figura 4-1: Funcionamiento de DPDK en el procesamiento de paquetes a nivel interno en comparación con la forma tradicional

4.1.3 Estructura

Algunas de las librerías más importantes que conforman DPDK son [7]:

→ **Environment Abstraction Layer (EAL)**. Es responsable de obtener acceso a recursos de bajo nivel como el hardware y el espacio de memoria. Proporciona una interfaz genérica que oculta los detalles del entorno de las aplicaciones y bibliotecas.

→ **Mempool**. Un pool de memoria es un asignador de un objeto de tamaño fijo. En DPDK, se identifica por su nombre y utiliza un manipulador mempool para almacenar objetos libres.

→ **MBUF**. Proporciona la capacidad de asignar y liberar búferes (mbufs) que pueden ser utilizados por la aplicación DPDK para almacenar búferes de mensajes. Estos búferes se almacenan en una mempool, utilizando la Librería Mempool.

Una estructura rte_mbuf generalmente lleva buffers de paquetes de red, pero en realidad puede ser cualquier dato (datos de control, eventos,...). La estructura de cabecera rte_mbuf se mantiene lo más pequeña posible y actualmente utiliza sólo dos líneas de caché, siendo los campos más utilizados los de la primera de las dos líneas de caché.

→ **Poll Mode Driver (PMD)**. Consiste en varias APIs, proporcionadas a través del controlador BSD que se ejecuta en el espacio de usuario, para configurar los dispositivos y sus respectivas colas. Además, un PMD accede a los descriptores de recepción (RX) y transmisión (TX) directamente sin interrupciones para recibir, procesar y entregar rápidamente los paquetes en la aplicación del usuario.

La **Interfaz NIC del Kernel** de DPDK (KNI) permite a las aplicaciones del espacio de usuario acceder al plano de control de Linux.

Los beneficios de usar el DPDK KNI son:

- Es más rápido que las interfaces Linux TUN/TAP existentes (eliminando las llamadas de sistema y las operaciones copy_to_user()/copy_from_user().

- Permite la gestión de puertos DPDK utilizando herramientas de red estándar de Linux como ethtool, ifconfig y tcpdump.

- Permite una interfaz con la pila de red del núcleo.

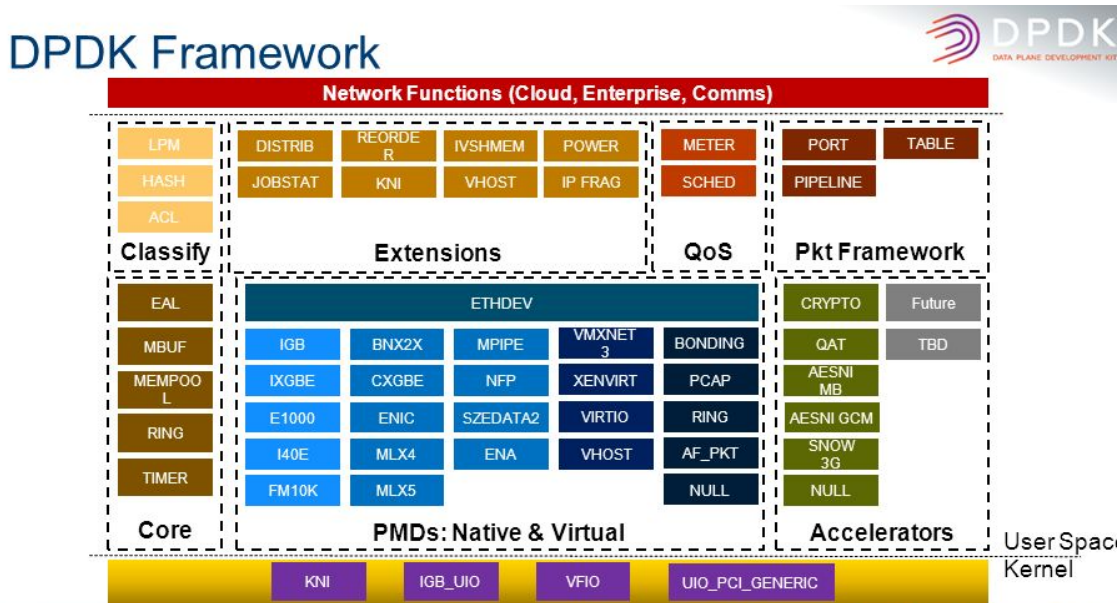

Figura 4-2: Estructura y librerias de DPDK

4.1.4 Configuración.

Inicialmente es necesario comprobar que nuestro sistema cumple con los requisitos mínimos explicados en el manual de DPDK (y en el Manual de Instalación en el Anexo 6.3). En nuestro caso, como hemos mencionado anteriormente, hacemos uso de máquinas virtuales para solucionar una de las mayores limitaciones, las tarjetas de red. De esta forma sacrificamos el rendimiento a favor de poder correr DPDK sin problemas y poder así, familiarizarnos con este entorno de desarrollo y su configuración.

Para empezar hay que descargar la versión de DPDK que vayamos a utilizar de su página oficial. Una vez descomprimida, y cumplidos los requisitos técnicos, podemos hacer uso de un script de configuración que se proporciona en los ficheros de DPDK, y que nos ayudará a simplificar todo el proceso. Antes de usar dicho script es necesario asegurarse de que las interfaces de red que vayan a ser utilizadas estén deshabilitadas, y hay que guardar la ruta a la carpeta donde está DPDK, en la variable RTE_SDK. Una vez hecho esto hay que ejecutar el script con permisos de superusuario.

En la figura 4-3 podemos ver que hay diferentes opciones para compilar el código en función de la plataforma, para insertar los diferentes módulos y unirlos a las interfaces de red, reservar las hugepages (imprescindible)… También nos da la opción de probar el correcto funcionamiento del sistema, mediante algunas aplicaciones de test.

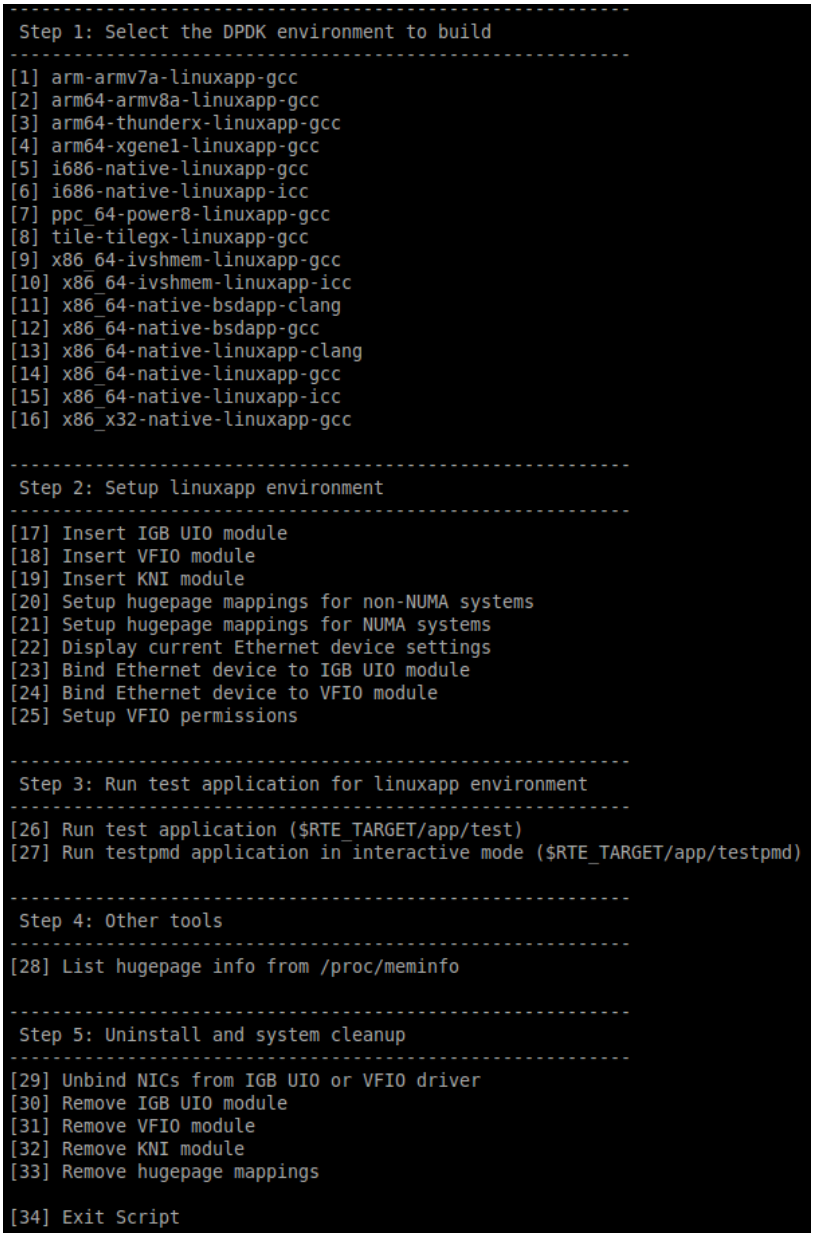

Figura 4-3: Menú del script de configuración de DPDK con todas las opciones posibles

Aunque este script es muy cómodo e intuitivo, me he decantado por seguir las instrucciones del manual de DPDK y configurarlo de manera manual como esta explicado en el Anexo, en el Manual de Instalación y Configuración. También se ha creado un script propio con todas las instrucciones necesarias para realizar la configuración de una manera óptima sin necesidad de perder el tiempo ejecutando los comandos uno a uno.

4.2 ANS (Accelerated Network Stack)

Esta pila TCP/IP correspondiente a las capas 3 (red) y 4 (transporte) del modelo OSI, nos va a proporcionar un espacio de usuario para poder ejecutarla junto con DPDK. Cuenta con una serie de funciones que nos permitirán implementar y probar la aplicación que queremos en las capas superiores.

4.2.1 Arquitectura y funcionamiento

Las librerías de ANS de las que haremos uso, cuentan con [9]:

- librte_ans: Librería estática de la pila TCP/IP. ANS usa dpdk mbuf, ring, memzone, mempool, timer, spinlock. así que no hay copia mbuf entre dpdk y ANS.
- librte_anssock: Librería de sockets para la aplicación, tampoco hay copia entre ANS y la aplicación.
- librte_anscli: Librería para la configuración de ruta/ip/link.
- Cli: Comando para configurar la pila ANS TCP/IP.7
- Example: Ejemplo de aplicación ANS.
- Test: Ejemplo de aplicaciones de ANS para probar la pila TCP/IP.

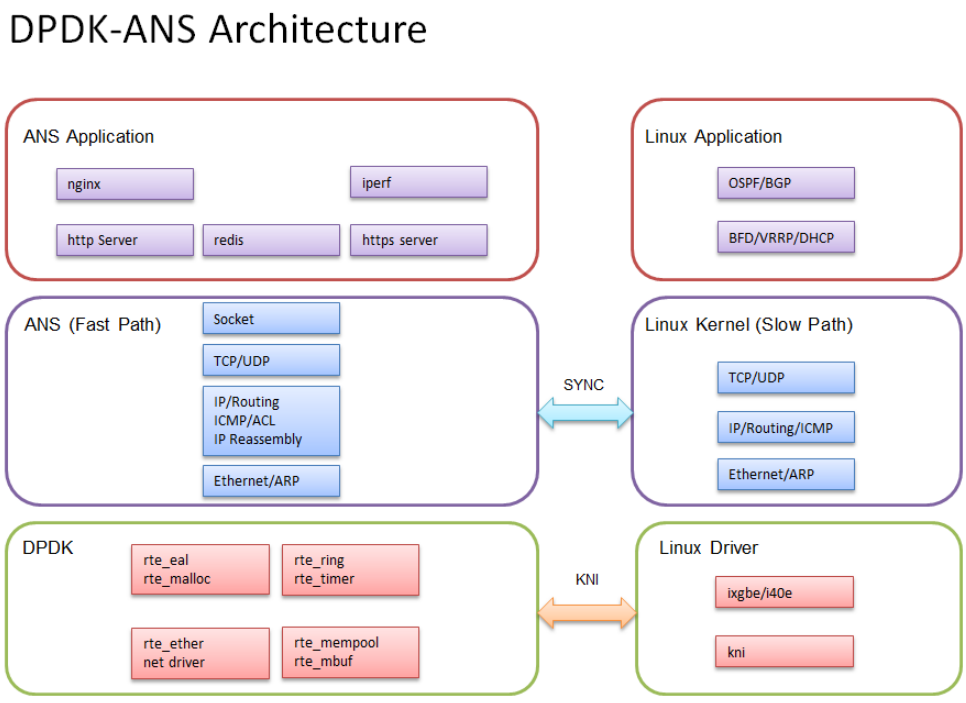

Figura 4-4: Comparación entre la arquitectura de ANS y la de Linux

Despliegue TCP

→ La NIC distribuye paquetes a diferentes lcore basados en RSS, por lo que el mismo flujo TCP se maneja en el mismo lcore.

→ Cada lcore tiene su propia pila TCP.

→ Si el proceso de la App sólo crea un socket de escucha, el socket de escucha sólo escucha en un lcore y acepta conexiones tcp desde el lcore, por lo que el número de proceso de la APP será mayor que el número lcore.

→ El proceso de la APP puede enlazar el mismo puerto si está habilitado como reuseport, y podría aceptar una conexión tcp por round robin [10].

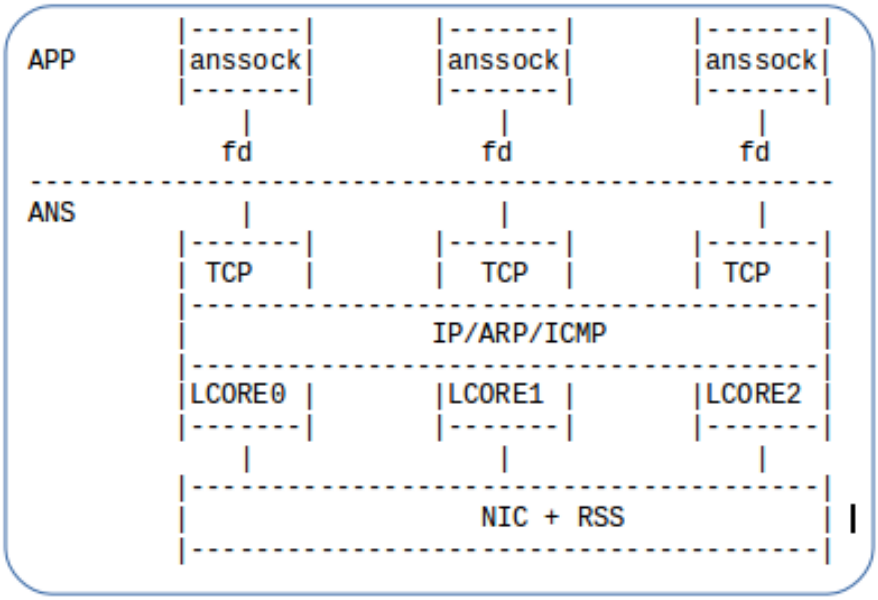

Figura 4-5: Despliegue de ANS. Estructura y conexión desde la NIC hasta el espacio de aplicación

4.2.2 Configuración

Inicialmente hay que comprobar que contamos con los requisitos necesarios para poder instalar y ejecutar ANS,y como ANS esta basado en DPDK muchos de estos requisitos serán los mismos. Estos se pueden encontrar en la página oficial de github de ANS, y también están incluidos en el Manual de Instalación en el Anexo.

Una vez tengamos la carpeta descargada y descomprimida, es necesario que la guardemos en otra carpeta junto a la de DPDK. Es muy recomendable que la versión de DPDK que descarguemos sea exactamente las misma que utiliza ANS, ya que sino pueden

surgir problemas. A continuación tenemos que configurar DPDK como hemos visto en el apartado 4.1.4, y cuando esté terminado deberemos situarnos en la carpeta de ANS y después de generar las librerías (./install_deps.sh), hay que exportar la ruta en la que se encuentran utilizando la variable RTE_ANS.

Después hay que ejecutar el proceso de ans con los parámetros EAL correspondientes dependiendo de cuantos cores queramos utilizar, cuantas interfaces de red haya.. (ejemplo: ./build/ans -c 0x4 -n 1 --base-virtaddr=0x2aaa2aa0000 -- -p 0x1 --config="(0,0,2)")

Finalmente, veremos que se nos ha creado una interfaz de red virtual (veth0).

```
#!/bin/bash
echo Compilando DPDK y ANS
export RTE_SDK=/home/luis_21/Escritorio/work/dpdk-stable-17.11.2
cd $RTE_SDK
export RTE_TARGET=x86_64-native-linuxapp-gcc
make install T=x86_64-native-linuxapp-gcc
sudo modprobe uio
sudo insmod $RTE_TARGET/kmod/igb_uio.ko
sudo insmod $RTE_TARGET/kmod/rte_kni.ko
sudo ./usertools/dpdk-devbind.py --force --unbind 00:03.0
sudo ./usertools/dpdk-devbind.py --bind=igb_uio 00:03.0
export RTE_ANS=/home/luis_21/Escritorio/work/dpdk-ans
cd $RTE_ANS
./install_deps.sh
cd ans
make
sudo sysctl -w kernel.randomize_va_space=0
./build/ans -c 0x2 -n 1 --base-virtaddr=0x2aaa2aa0000 -- -p 0x1 --config="(0,0,1)" --enable-kni --enable-ipsync
```

Figura 4-6: Script de compilación y ejecución de DPDK y ANS

En este script primero se exportan las variables necesarias y se compila DPDK. A continuación, se cargan los módulos necesarios y se unen los drivers con las interfaces de red. Después se compila ANS y se ejecuta ajustando los parámetros EAL para decidir cuantos cores y puertos, y cuales utilizamos.

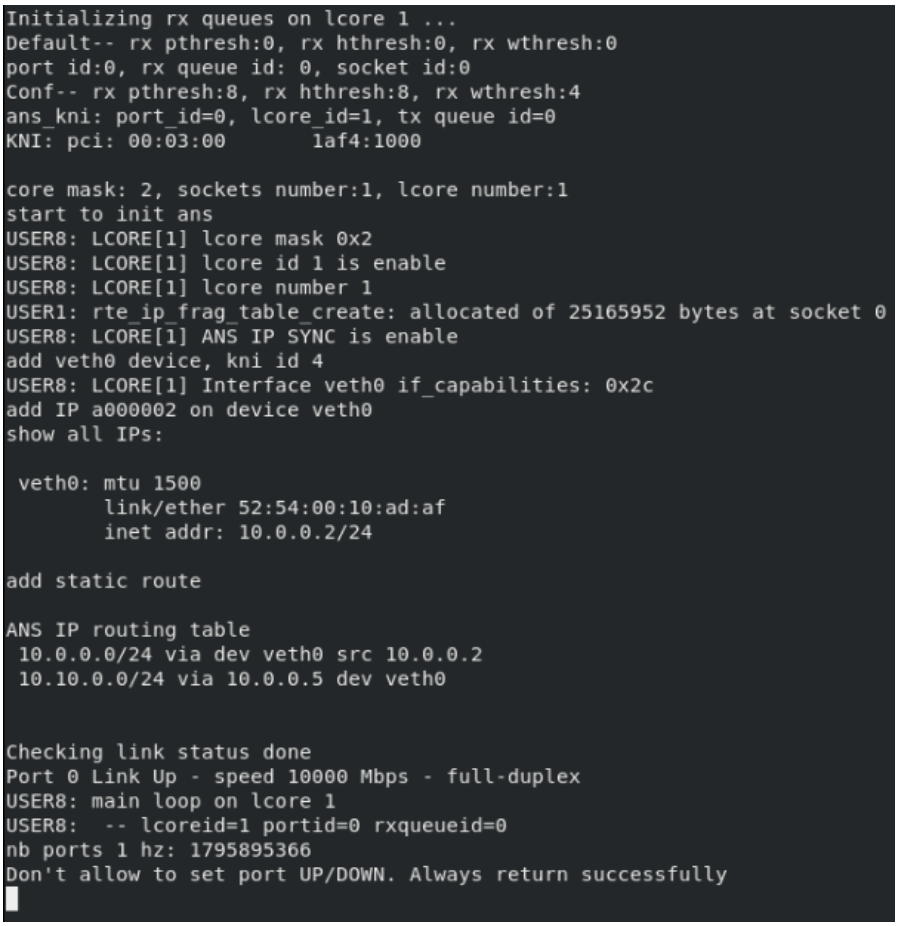

Figura 4-7: Terminal 1 después de ejecutar ANS

Para configurar ANS tenemos que hacer uso de la librería anscli (./build/anscli). Anscli nos va a permitir diversas opciones de configuración como:

→ Configurar la IP de la interfaz virtual (añadir, borrar…)

→ Configurar la ruta.

→ Ver el estado del link.

→ Muestra los datos de la IP.

Al igual que con la compilación y configuración de DPDK, para compilar y ejecutar ANS también se ha creado un script con el propósito de ahorrar tiempo y automatizarlo.

Sin embargo hay algunos comandos que tenemos que meter manualmente y que dependen de nuestra red. Estos comandos son:

- Añadir una IP para identificar nuestra interfaz, ya que se añade una por defecto en ANS que hay que modificar.

```
Abriendo anscli..
ans> ip addr add 192.168.0.101/24 dev veth0
Add IP address successfully
ans>
```

Figura 4-8: Comando de configuración de la IP

- Añadir una ruta hasta el router para poder comunicarnos fuera de nuestra red local.

```
ans> ip route add 0.0.0.0/0 via 192.168.0.1
Add route successfully
ans>
```

Figura 4-9: Comando de configuración de la ruta

Para comprobar que todo esta bien:

```
ans> ip addr show

 veth0: mtu 1500
   link/ether 52:54:00:10:ad:af
   inet addr: 10.0.0.2/24
   inet addr: 192.168.0.101/24
ans> ip route show

ANS IP routing table
 0.0.0.0/0 via 192.168.0.1 dev veth0
 10.0.0.0/24 via dev veth0 src 10.0.0.2
 10.10.0.0/24 via 10.0.0.5 dev veth0
 192.168.0.0/24 via dev veth0 src 192.168.0.101
ans> ip link show

 veth0: port 0 state UP speed 10000Mbps full-duplex mtu 1500
   link/ether 52:54:00:10:ad:af
   IPV4 checksum offload disable
   UDP checksum offload enable
   TCP checksum offload enable
   TCP TSO offload enable
   Max allowed number of segments: 0
   RX packets:353 errors:0 dropped:31
   TX packets:103 errors:0 dropped:0
ans>
```

Figura 4-10: Comandos para ver el estado de la ruta y la interfaz virtual creada

Una vez hayamos ejecutado el proceso principal de ANS (Figura 4-6), es necesario abrir otro terminal para llevar a cabo la configuración.

4.3 Libtrading

4.3.1 Introducción

Como se ha comentado en el apartado 2.6, el protocolo FIX nos va a permitir comunicarnos con otras entidades, para compartir información de los mercados financieros. Nuestro objetivo es desarrollar un cliente que haga la función de comprador o trader que se va a comunicar con un servidor, que haga las veces de contraparte proporcionándonos precios demo. Simularemos así una comunicación real entre un trader y el mercado.

Este cliente corresponde a las capas 5, 6 y 7 (aplicación) del modelo OSI. Antes de realizar la simulación se ha tenido que invertir bastante tiempo en la familiarización con estas librerías y sus funciones, y en la modificación de las mismas para integrarlas con las capas inferiores y añadir algunas funcionalidades extra.

Para el desarrollo del cliente se han seguido una serie de pasos hasta llegar a la versión final:

1. Familiarización con las estructuras de las variables y las funciones para poder utilizar el protocolo FIX y entender como funciona.
2. Modificación de algunos archivos de *libtrading* para adaptarlos a los requerimientos pedidos por nuestro proveedor de precios (servidor).
3. Estructurar y programar el diálogo entre cliente y servidor haciendo uso de las funciones proporcionadas por ANS para realizar la conexión entre sockets.

4.3.2 Familiarización con el protocolo FIX

Para empezar a trabajar con este protocolo lo primero de todo es crear e iniciar una sesión FIX correctamente. Después el usuario podrá enviar y recibir mensajes de forma inmediata. Por lo tanto un ciclo de trabajo de FIX consta de tres etapas: crear e iniciar la sesión, realizar las operaciones necesarias y destruir la sesión.

- Para ejecutar una nueva sesión basta con invocar una función que acepte una configuración deseable como argumento de entrada y devuelva una nueva sesión en caso de éxito. La estructura de configuración consiste en un conjunto de parámetros particulares aplicables a la sesión recién creada. Entre ellos se encuentran el descriptor de sockets utilizado para la comunicación, SenderCompID y

TargetCompID, la contraseña… Todos ellos parámetros necesarios para establecer la conexión con el proveedor.

- La base de este protocolo son los mensajes FIX. Estos mensajes consisten en campos FIX que determinan completamente el tipo de mensajes y se utilizan para transferir datos entre contrapartes. Hay algunos campos obligatorios que deben estar en cada mensaje FIX transferido. Estos campos se pueden direccionar directamente dentro de la estructura fix_message. Todos los campos restantes se mantienen en una simple matriz lineal.

```
struct fix_message {
        /*
         * These are required fields.
         */
        const char                      *begin_string;
        unsigned long                   body_length;
        const char                      *msg_type;
        const char                      *sender_comp_id;
        const char                      *target_comp_id;
        unsigned long                   msg_seq_num;

        unsigned long                   nr_fields;
        struct fix_field                *fields;
};
```

Figura 4-11: Estructura de un menaje FIX dentro de *libtrading*

- Hay dos acciones principales que se pueden realizar con los mensajes: transmitir y recibir. Para recibir un mensaje se debe utilizar la función fix_session_recv(). Para enviar mensajes *libtrading* proporciona un conjunto de funciones de diferentes tipos. Estas funciones son esencialmente envolturas alrededor de la función general fix_session_send().

Vamos a profundizar un poco más en la estructura de los mensajes FIX, dado que es vital que no haya ningún fallo en el dialecto utilizado a la hora de comunicarnos con el proveedor.

El campo FIX es sólo un par de Tag y Value que aparece en el mensaje como "Tag=Value" seguido del trailer estándar de FIX. Para poder analizar un valor correctamente debemos saber a qué tipo pertenece. Podría ser uno de los tipos de datos definidos en la especificación FIX, es decir, char, int, float, string, etc. Una vez que se conoce el tipo de datos, se puede analizar todo el campo FIX.

Podemos decir que una etiqueta en particular es conocida si podemos analizar su valor correspondiente. Un conjunto de Tags conocidos se llama dialecto. En otras palabras, un dialecto es el conjunto de campos FIX que podemos analizar correctamente [11].

Son necesarios diferentes dialectos, ya que en la vida real existen diferentes versiones de FIX a las cuales nos tenemos que adaptar si queremos que la comunicación sea correcta. También tenemos la libertad de poder desarrollar nuestro propio dialecto en vez de utilizar uno de los ya existentes. En nuestro caso se ha utilizado un dialecto ya existente con la versión FIX.4.4. Sin embargo, este dialecto no contenía todos los 'tag' que se necesitaban para comunicarse con el proveedor (PrimeXM), por lo que se han tenido que añadir manualmente algunas etiquetas (tag) de acuerdo a los requerimientos especificados por PrimeXM.

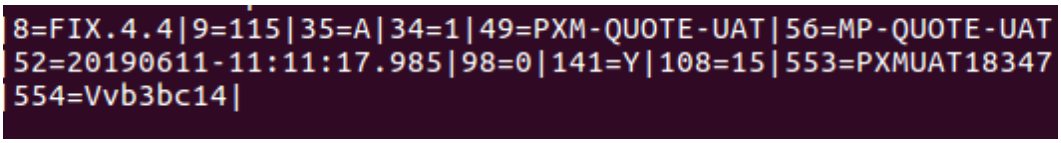

Figura 4-12: Menaje FIX de logon. Se aprecian los diferentes "tag=value" separados por el simbolo '|'

4.3.3 Integración de ANS y *libtrading*

Conociendo las opciones que nos ofrece *libtrading* y sabiendo manejar las diferentes funciones con las que se cuenta, es hora de conectar esta librería con las capas inferiores que hemos ido integrando a lo largo del desarrollo, para poder aprovecharnos de las optimizaciones en el procesado de paquetes. Para ello ANS dispone de una serie de funciones, situadas en al carpeta librte_anssock, similares a las funciones de manejo de sockets proporcionadas por la librería <sys/socket.h>. Las funciones proporcionadas por ANS se sirven de DPDK para conseguir tiempos menores a la hora de ejecutarse.

Para poder realizar la conexión entre libtrading y ANS, aparte de modificar el makefile para incluir las librerías correspondientes, se han modificado las funciones originales de los sockets, por las correspondientes de ANS, que tienen el prefijo 'anssock_'.

Dentro de read-write.c (en libtrading-master/lib/), se han sustituido las funciones:

I. ***sendmsg(fd, &msg, flags)*** → ***anssock_writev(fd, iov, msg.msg_iovlen)***.

II. ***io_recv = &recv*** → ***io_recv = &anssock_recv.***

III. ***read(fd, buf, count) → anssock_read(fd, buf, count).***

IV. ***write(fd, buf, count) → anssock_write(fd, buf, count).***

V. ***writev(fd, iov, iovcnt) → anssock_writev(fd, iov, iovcnt).***

4.3.4 Diálogo cliente-servidor

Para crear el cliente primero se han utilizado algunos programas de prueba que venían en los archivos de ANS y de *libtrading.* ANS dispone de ejemplos de programas cliente y servidor tanto con DPDK como normales. Y *libtrading* dispone de un ejemplo de un servidor y de un cliente que se comunican con las funcionalidades básicas del protocolo FIX, como son: crear una sesión, hacer logon, enviar ordenes y recibirlas, y hacer logout, entre otras.

Se ha hecho uso tanto del cliente de *libtrading* como del cliente de ANS para programar un nuevo cliente FIX que nos permita utilizar las funciones de ANS y de la estructura de *libtrading* y poder comunicarnos así con el servidor de prueba proporcionado por *libtrading.* Esto se ha hecho con el objetivo de desarrollar nuestro programa lo mejor posible, detectando los errores en un entorno local de prueba, antes de enfrentarlo al servidor de verdad que nos proveerá de precios demo.

El cliente desarrollado se ejecuta en una máquina virtual con el sistema operativo Fedora 28, y el servidor se ejecuta en el host, con sistema operativo Ubuntu 16.04.

Como hemos dicho antes se ha integrado el cliente con las funciones de conexión de sockets proporcionadas por ANS, de esta manera se consigue disminuir el tiempo que se tarda en enviar y recibir ordenes y precios. En el servidor sin embargo no se ha modificado nada con respecto a los sockets, solamente se han modificado aquellos aspectos de la comunicación del protocolo FIX necesarios para crear el diálogo deseado entre trader y mercado.

Los aspectos principales de la comunicación FIX que se han desarrollado en el entorno de pruebas son [12]:

1. Logon por parte del cliente y repuesta de logon por parte del servidor.
2. Envío de ordenes individuales por parte del cliente.
3. Procesado de dichas ordenes por parte del servidor.
4. Logout por parte del cliente y respuesta de logout del servidor.

5 Integración, pruebas y resultados

En este capítulo se expone el proceso seguido a la hora de integrar el cliente final en el entorno de pruebas, la realización de las mismas y los resultados obtenidos. Para ello vamos a partir del apartado 4.3.4 en el que hemos explicado el dialogo creado entre el cliente y el servidor y las pruebas a nivel local realizadas en un solo ordenador con el entorno virtualizado.

5.1 Integración y pruebas

En el apartado 4.3.4 hemos desarrollado las funcionalidades básicas del cliente, y para comprobar su correcto funcionamiento se ha hecho uso de la herramienta wireshark. Se ha capturado el tráfico dentro de nuestro entorno local de pruebas, programando a nuestro cliente para que haga logon y logout simplemente.

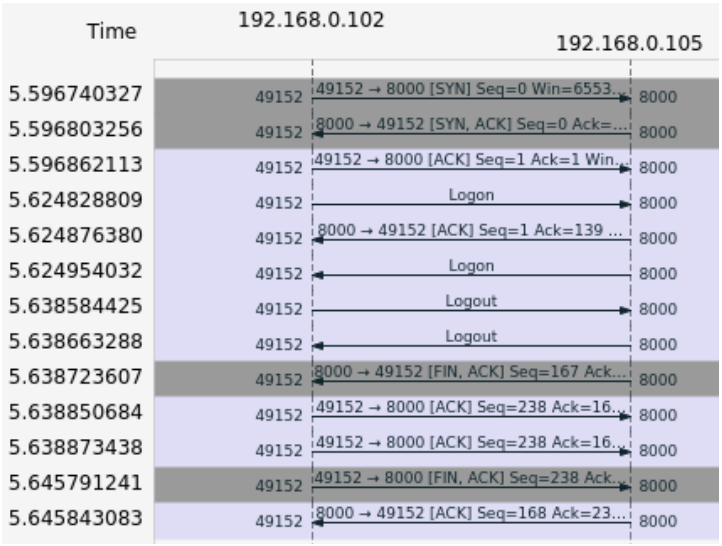

Figura 5-1: Traza del primer diálogo de prueba

Como apreciamos en la Figura 5-1,primero se comprobó el diálogo correcto entre el cliente y el servidor mediante una breve conversación en la que se hacia logon y logout esperando las correspondientes respuestas. La figura muestra la interacción que se realizó en el entorno de pruebas entre el cliente en la máquina virtual (IP: 192.168.0.102) y el servidor en el host (IP: 192.168.0.105).

Teniendo el programa ya desarrollado y funcionando correctamente tras las diversas conexiones y pruebas realizadas entre el cliente (ejecutado en la máquina virtual) y el

servidor (ejecutado en el host), ahora toca conectar el cliente con el servidor de PrimeXM que se nos ha proporcionado . Para ello se nos ha facilitado una dirección IP, un puerto, un usuario y una contraseña. Una vez introducidos los datos en el código se ha conectado con el servidor siguiendo el mismo esquema que en la Figura 5-1. Después de comprobar la conectividad y verificar que el servidor respondía correctamente, se ha establecido un diálogo más completo en el que aparte de hacer logon y logout se envía una solicitud de precios de mercado. Una vez hecho esto el servidor nos envía de forma continua precios del símbolo al que nos habíamos subscrito, en nuestro caso, el EUR/USD.

Figura 5-2: Traza de la solicitud de precios

En la Figura 5-2 se aprecia la comunicación entre nuestro cliente FIX (IP: 192.168.0.101) y el servidor externo (IP: 54.93.61.215) que hace las veces de mercado. Después de hacer el logon y recibir una respuesta por parte del servidor, el cliente envía un 'MarketDataRequest', esto es un mensaje para subscribirse a un símbolo (EUR/USD) y

que nos empiecen a llegar mensajes de precios sobre dicho símbolo. Así una vez el servidor recibe este mensaje comienza a enviarnos mensajes ‘MassQuote’ que contienen la información sobre el símbolo subscrito en tiempo real (BID, ASK, cantidad…). Cuando se quiere finalizar la conexión el cliente hace un logout con su correspondiente respuesta del servidor.

Teniendo la comunicación entre cliente y servidor funcionando correctamente, ahora solo habría que procesar los paquetes y extraer la información que nos resultase más útil, pudiendo crear así una estrategia para invertir.

No se ha creado una estrategia de inversión debido a la falta de conocimiento sobre inversión, la falta de tiempo y que no era el objetivo de este TFG. Sin embargo, se ha programado una pequeña funcionalidad para comprobar que la información se extrae de los mensajes recibidos correctamente. Consiste simplemente en detectar cuando están bajando los precios de forma continua y avisar mediante una notificación por pantalla al programa. Esta notificación podría ser sustituida por una orden de compra o de venta correspondiente si se tratase de un caso real de una estrategia de inversión.

5.1.1 Medida de la latencia

El objetivo de este TFG residía en disminuir el tiempo de procesado de las ordenes de un motor FIX. Dicho motor corresponde con nuestro cliente FIX. Para obtener los tiempos de procesado de las ordenes, se ha medido de forma independiente en el cliente, el tiempo de las dos operaciones esenciales que necesitamos, enviar y recibir.

A la hora de medir el tiempo de envío de una orden se han utilizado diferentes tipos de mensajes (logon, logout, MarketDataRequest) que el cliente envía al servidor durante la conexión. Para medir el tiempo al recibir, se han medido los mensajes que el servidor nos enviaba, es decir, aquellos que contienen el precio de los pares de divisas (MassQuote).

Para medir los tiempos se han utilizado las funciones de C clock() y clock_gettime(). Al enviar, las medidas se han realizado desde que se da la orden de envío en el cliente hasta que dicha función devuelve algún resultado correcto, es decir, hasta que el mensaje se envía correctamente y sale por nuestra NIC. Y al recibir, las medidas se han realizado desde que estamos esperando recibir un mensaje hasta que lo recibimos (entra por nuestra NIC) y lo tenemos disponible para procesarlo en el cliente.

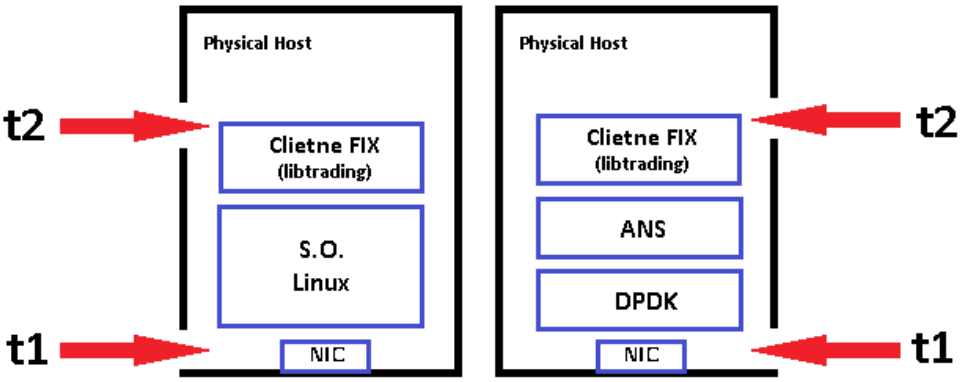

Figura 5-3: Esquema de la medición de tiempos. Al recibir, se mide el tiempo desde t1 hasta t2, y para enviar al revés, desde t2 hasta t1

Cabe destacar que la optimización no solo se encuentra en la utilización de ANS y DPDK. El cliente FIX ha sido programado en lenguaje C, un lenguaje de muy bajo nivel que nos permite alcanzar velocidades de procesamiento muy altas en comparación a otros lenguajes de más alto nivel.

Para poder corroborar la optimización del tiempo medido, se ha desarrollado un cliente exactamente igual que el principal, pero utilizando las funciones de los sockets incluidas en las librerías del sistema linux por defecto (Figura 5-3). Es decir, se ha creado otro cliente que no está basado en las librerías de DPDK ni ANS, se ha probado su conectividad con el servidor de PrimeXM y se han medido los tiempos de envío y de recibo de mensajes de igual manera que con el otro cliente FIX.

Sin embargo, tanto el cliente FIX optimizado como el normal, hacen uso de la librería de *libtrading*, por lo que la comparación de los tiempos hace referencia únicamente al uso de ANS y DPDK como herramientas para la optimización.

5.2 Resultados

Se han tomado medidas de los tiempos tanto al enviar los mensajes como al recibirlos. Se han representado los tiempos de 100 ordenes/mensajes al ser recibidos y 50 al ser enviados.

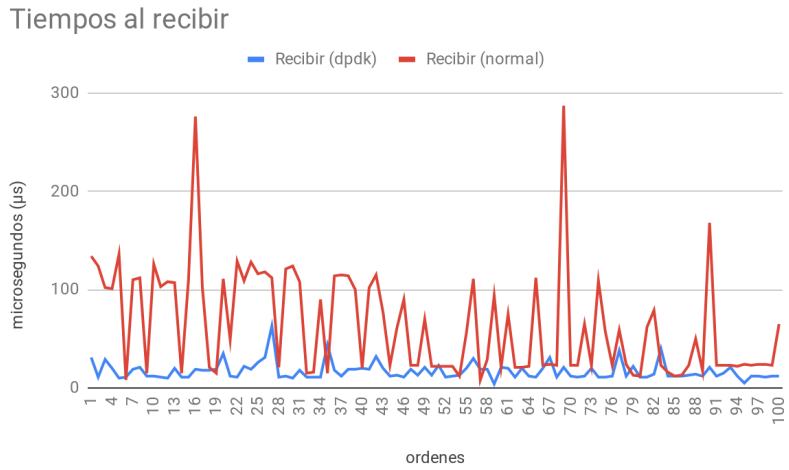

Figura 5-4: Gráfica de lineas de los tiempos al recibir mensajes de MassQuote

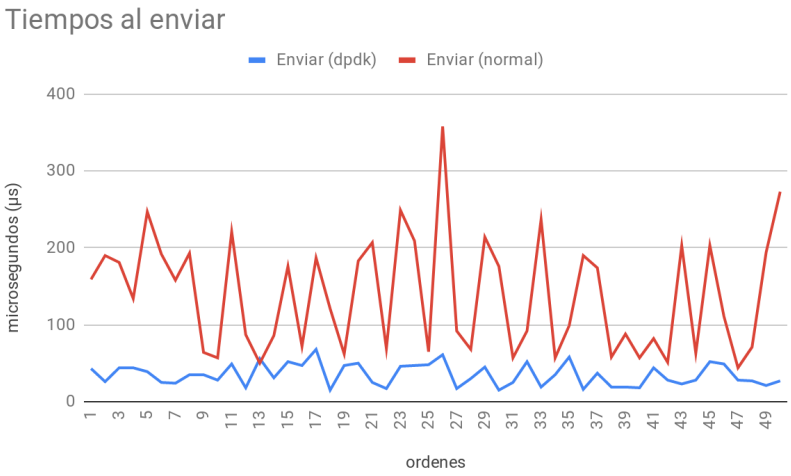

Figura 5-5: Gráficas de lineas de los tiempos al enviar mensajes de logon, logout y MarketDataRequest

En las figuras 5-4 y 5-5 se aprecian claramente las diferencias de tiempos al medir utilizando el cliente FIX con la pila de protocolos implementada en este TFG y el cliente FIX (similar al anterior en la capa de aplicación) con la pila de protocolos común del S.O. Linux.

También se ha calculado la mediana de todos los mensajes en cada caso, para tener una mejor aproximación.

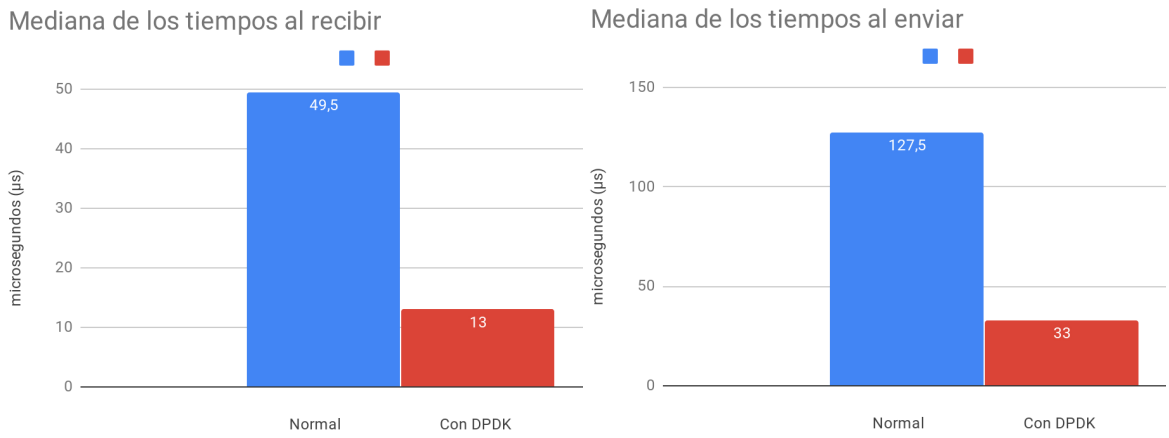

Figura 5-6: Gráficas comparativas de los tiempos medidos al enviar y al recibir los diferentes mensajes, utilizando la tecnología de DPDK y sin utilizarla

Como podemos observar en la gráfica de la Figura 5-6, se ha conseguido optimizar el tiempo tanto a la hora de enviar y como de recibir mensajes en un factor de 4. También se aprecia que la función al enviar tarda el doble de tiempo aproximadamente, que al recibir en ambos casos.

Es importante destacar que estos tiempos se han obtenido en un entorno virtualizado con SO Fedora 28, y en un ordenador Sony VAIO, Intel core i3. Es decir, aunque los tiempos son del orden de microsegundos, y se ha conseguido una optimización del procesamiento de los paquetes de red 4 veces mayor, en un entorno real con una tarjeta de red apropiada y compatible se podrían conseguir tiempos aún menores.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

El aumento de la velocidad y la disminución de la latencia son exigencias cada vez más presentes en el mundo de las telecomunicaciones. Año tras año se producen avances tanto en hardware con en software que nos permiten aumentar las velocidades de procesamiento y de transmisión de información.

No en vano se dice que estamos en la era de la información. Y es que hoy en día el que tiene la información tiene el "poder", sobre todo en el mundo de los mercados financieros.

Por esto, este TFG se ha centrado en el desarrollo de una pila de protocolos y un software que nos permitan reducir la latencia a la hora de operar en los mercados financieros, en concreto los de FOREX.

Se ha conseguido reducir en un factor de 4 el tiempo que se tarda en recibir, procesar y enviar las ordenes desde una de las contrapartes.

Estamos en un momento en el que las operaciones de trading las realizan programas informáticos mediante algoritmos que operan a velocidades increíblemente altas (del orden de pocos microsegundos y menos). Por eso, dentro de este ámbito una optimización de tal calibre como la desarrollada en este TFG, puede marcar una diferencia abismal entre los diferentes programas que operan en los mercados financieros. Estos programas no solo serían capaces de aumentar el número de operaciones realizadas en un corto período de tiempo, sino que podrían disponer de la información de los mercados y las cotizaciones mucho antes, lo cual aumenta notablemente la fiabilidad y efectividad de cualquier algoritmo utilizado.

6.2 Trabajo futuro

Con este TFG no solo se ha conseguido una optimización aplicable a las operaciones en los mercados financieros. Al desarrollar una pila de protocolos optimizada, tenemos a nuestra disposición una serie de librerías adaptables a cualquier uso que se les quiera dar.

Partiendo del trabajo realizado y el conocimiento obtenido en el desarrollo de este TFG, se plantean diversos objetivos como trabajo futuro:

- Optimización del cliente desarrollado mejorándolo y añadiéndole más funcionalidades a la hora de operar en el mercado.
- Desarrollo de un algoritmo complejo de inversión utilizando la pila de protocolos aquí desarrollada.
- Implantación de inteligencia artificial para el análisis de los mercados financieros utilizando las librerías de DPDK, ANS y libtrading.

Referencias

[1] Intel Corporation. DPDK home page. https://www.dpdk.org/ , Octubre 2018.

[2] Red Hat. Fedora home page. https://getfedora.org/es/ , Octubre 2018.

[3] Wiki, Linux Virtualization, “VEPA, Bridge and private mode”, https://virt.kernelnewbies.org/MacVTap?action=fullsearch&context=180 , Octubre 2018

[4] KVM. “Configuring Guest Networking”. https://www.linux-kvm.org/page/Networking , Octubre 2018.

[5] QEMU home page. https://www.qemu.org/ , Octubre 2018.

[6] Intel Corporation. DPDK download page. http://core.dpdk.org/download/ , Octubre 2018.

[7] Intel Corporation. DPDK documentation page. http://core.dpdk.org/doc/ , Octubre 2018.

[8] Mr. Hrishikesh Kulkarni, Mr. Sachin Agrawal, Mr. Rohan Pore, Miss. Priti Andhale, Mrs. Nilam Patil, “A survey on TCP/IP API stacks based on DPDK”, IJARIIE-ISSN(O)-2395-4396, Vol-3 Issue-2, 2017.

[9] ANS GitHub repository. https://github.com/ansyun/dpdk-ans , Diciembre 2018.

[10] ANS home page. http://www.ansyun.com/ , Febrero 2019.

[11] Libtrading GitHub repository. https://github.com/libtrading/libtrading , Enero 2019.

[12] PrimeXM. “PrimeXM FIX 4.4 Trading API Specification”. V1.5.7 . Marzo 2019.

Anexos

6.3 Manual de instalación y configuración

Maquina Virtual (QEMU/KVM)

Virtualización: KVM

SO: Fedora 28 , 64bits

Configurar la conexión de la máquina en modo 'bridge'.

Para empezar:

Una vez tengamos las maquina virtual encendida, vamos a descargar tanto DPDK (la misma versión que se especifica aquí) como ANS, y al descomprimirlos los vamos a guardar en un mismo directorio (por ejemplo: 'work').

IMPORTANTE: mirar la versión de DPDK que soporta ANS, ya que van actualizándola y es muy recomendable que sea la misma.

- Descargar DPDK:

→ wget http://dpdk.org/rel/dpdk-18.11.tar.xz

→ xz -d dpdk-18.11.tar.xz

→ tar xvf dpdk-18.11.tar

- Descargar ANS:

→ git clone https://github.com/ansyun/dpdk-ans.git

DPDK

- Pre-requisitos:

Hay que habilitar el High Precision Timer (HPET), ya que es usado por DPDK. Para ello vamos a la pantalla de la BIOS del sistema y la ruta sería: Advanced -> PCH-IO Configuration -> High Precision Timer -> (Change from Disabled to Enabled if necessary).

Para comprobar que HPET esta habilitado podemos utilizar este comando:

→ grep hpet /proc/timer_list

Si no devuelve nada, debemos habilitar HPET en la BIOS y reiniciar el sistema.

Nota: Si el arranque seguro UEFI está habilitado, el núcleo de Linux puede no permitir el uso de UIO en el sistema.

Paquetes necesarios:

- gcc: versions 4.9 or later is recommended

- libc headers, often packaged as gcc-multilib (glibc-devel.x86_64 for 64-bit compilation on Intel architecture)

- Linux kernel headers or sources required to build kernel modules. (kernel-devel.x86_64)

- Library for handling NUMA (Non Uniform Memory Access).

→ yum install numactl-devel

→ yum install libpcap-devel

→ yum install kernel-devel

Si da error igb_uio, instalar:

→ yum inatall kernel-devel-xxxx , siendo xxxx = lo que devuelve el comando:

→ uname -r

→ yum install elfutils-libelf-devel

Paquetes adicionales (para Intel):

- glibc.i686, libgcc.i686, libstdc++.i686 and glibc-devel.i686

Exportar variables:

→ export RTE_SDK=/.../directorio_de_dpdk (poner la ruta de tu ordenador y el nombre de la carpeta en la que está dpdk)

→ export RTE_TARGET=x86_64-native-linuxapp-gcc

Reserve Hugepages:

- Para configurar el tamaño de las hugepages:

  → sudo vi /etc/sysctl.conf

- Agregar al final del archivo:

  → vm.nr_hugepages=256

- Para montar las hugepages al inicio:

  → sudo vi /etc/fstab

- Agregar al final del archivo:

  → huge /mnt/huge hugetlbfs defaults 0 0

  → sudo mkdir /mnt/huge_

  → sudo chmod 777 /mnt/huge_

- Mirar estado de las hugepages:

  → grep -i huge /proc/meminfo

Para compilar:

→ make config T=x86_64-native-linuxapp-gcc

→ make install T=x86_64-native-linuxapp-gcc

Cargar modulos del Kernel (para entorno virtualizado)

→ sudo modprobe uio

→ sudo insmod $RTE_TARGET/kmod/igb_uio.ko

→ sudo insmod $RTE_TARGET/kmod/rte_kni.ko

Unir puerto de red

Se tiene que utilizr igb_uio, ya que vfio no soporta virtualización.

(use --force if DEVICE is active) (DEVICE = 0000:xx:xx.x ej: 00:03.0)

Los DEVICES de tu sistema se pueden ver con –status; el comando de abajo.

→ sudo ./usertools/dpdk-devbind.py --status

Para separar (desunir):

→ sudo ./usertools/dpdk-devbind.py --force --unbind DEVICE

Para unir:

→ sudo ./usertools/dpdk-devbind.py --bind=igb_uio DEVICE

ANS

Pre-requisitos:

Crear directorio (por ejemplo: 'work') y guardar en él tanto los archivos de ANS como de DPDK.

Exportar variable:

→ export RTE_ANS=/home/.../dpdk-ans

Situarse en el directorio de ANS y modificar el script install_deps.sh si sale algun error, ya que la arquitectura del procesador del ordenador utilizado debe ser compatible con las soportadas por ANS. En nuestro caso se ha modificado en el script: native = westmere). Después ejecutar el script:

→ ./install_deps.sh

Compilar ANS:

→ cd ans

→ make

Para evitar errores por usar procesos secundarios (sockets):

→ sudo sysctl -w kernel.randomize_va_space=0

Correr ANS: (por ejemplo: con 2 cores 0 y 1, corriendo en el 1. Para mas información, mirar documentación sobre los parámetros EAL)

→ ./build/ans -c 0x2 -n 1 --base-virtaddr=0x2aaa2aa0000 -- -p 0x1 --config="(0,0,1)"_

Abrir otro terminal y situarnos en el directorio de ANS (→ cd $RTE_ANS):

→ cd cli

→ ./build/anscli

- Cuando estemos dentro de anscli podemos manejar las ip, mirar las rutas... (Mirar documentacion de anscli)

- Configurar la ip, añadimos una que este en nuestro rango:

  ans> ip addr add 192.168.0.101/24 dev veth0

- Configurar la ruta:

ans> ip route add 0.0.0.0/0 via 192.168.0.1

Libtrading

Descargar de GitHub el codigo https://github.com/libtrading/libtrading y sustituir los ficheros subidos aquí: https://github.com/luis0021/FIX-engine-over-ANS por los que contiene el programa original.

Se ha modificado el makefile para poder utilizar con libtrading las funciones de ANS para los sockets.